VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Uživateľské rozhranie s MS Kinect

User Interface with MS Kinect

2013 Pavol Kubjatko

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Pavol Kubjatko**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Uživatelské rozhranní s MS Kinnect
User Interface with MS Kinnect

Zásady pro vypracování:

S příchodem technologie Kinnect firmy Microsoft se otevřela možnost ovládání počítače pohybem končetin, těla či gesty. Cílem diplomové práce je návrh a prototypová implementace uživatelského rozhranní aplikace ovládaného pomocí výše uvedeného zařízení.
Práce bude obsahovat tyto body:
1. Prostudujte možnosti zařízení MS Kinnect a jeho programování.
2. Navrhněte demostrační aplikaci a její uživatelské rozhranní.
3. Aplikaci implementujte.
4. Popište testy aplikace a vyhodnoťte je.

Seznam doporučené odborné literatury:

Jarrett Webb and James Ashley. Beginning Kinect Programming with the Microsoft Kinect SDK. Apress. 2012. ISBN 978-1430241041.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Doc. Mgr. Jiří Dvorský, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 6. apríl 2013 .....

Rád by som poďakoval všetkým, ktorý mi s touto prácou pomohli, pretože bez nich by táto práca nevznikla.

Abstrakt

Práca poukazuje na výhody a nevýhody použitia kinectu. Popisuje možnosti programovania kinectu. Ukazuje ake gestá sa dajú použiť pri použití algoritmov a, ktoré nie. Popisuje tvorbu jednoduchej ukážkovej aplikácie a taktiež popisuje hotové naprogramované aplikácie a ich gesta. Na konci práce sú testy a ich vyhodnotenie.

Kľúčové slova: Kinect, UserControl, Gestá, Joint, WPF

Abstract

The work highlights the advantages and disadvantages of using Kinect. Describes Kinect programming options. The work will show how gestures can be used when using algorithms and which are not. Describes the creation of a simple sample application also describes a fully programmed applications and their gestures. At the end of the work there are tests and their evaluation.

Keywords: Kinect, UserControl, Gestures, Joint, WPF

Zoznam použitých skratiek a symbolov

UI	–	User Interface
IR	–	Infra Red
SDK	–	Software Development Kit
WPF	–	Windows Presentations Foundations
XAML	–	Extensible Application Markup Language
RSS	–	Rich Site Summary
UC	–	User Control

Obsah

Zoznam tabuliek

Zoznam obrázkov

Zoznam výpisov zdrojového kódu

1 Úvod

Táto práca vznikla za účelom zistenia, aké sú možností programovanie kinectu pre Windows. V prvej kapitole sa zoznámime s kinectom. Čo to vlastne kinect je. Prečo vznikol. K čomu kinect slúži a z čoho je zložený. V ďalšej kapitole sa bližšie zoznámime s možnosťami SDK kinect pre Windows. Čo všetko nám môže poskytnúť. Zistíme čo sa rozumie pod pojmom gesto v kinecte, aké gesta sú bežne používané a pomocou čoho sa dajú gestá programovať. V ďalšej kapitole si ukážeme jednoduchú aplikáciu tutoriál a celú ju popíšeme. V ďalších kapitolách si popíšem dve funkčné aplikácie s použitím kinectu. A na konci vyhodnotíme testy.

2 O Kinecte

2.1 Zoznámenie

Kinect je pohybový senzor, ktorý je určený na hĺbkové snímanie jednej alebo viacerých osôb. Pomocou infračervených senzorov dokáže rozpoznávať 3D objekty. Je naprogramovaný aby dokázal rozpoznávať, či sa jedná o osoby. Ak softvér vyhodnotí že sa jedná a človeka začne ho sledovať. Tento algoritmus nie je dokonalý, zo skúsenosti sa dá povedať že si dokáže pomýliť šatník s oblečením prehodeným cez dvere.

Kinect poskytuje štyri typy prichádzajúcich dát. Video stream je získavaná z RGB kamery s maximálnym rozlíšením 1280x960 pixelov. Hĺbkový(depth) stream je získavaný z IR senzora s maximálnym rozlíšením 640x480 pixelov. Sú to dáta s, ktorými kinect naozaj pracuje. Pomocou knižníc vytvorených Microsoftom dokáže rozpoznať či sa jedná o skeleton(osoba), čo je tretí typ streamu. Kostra sú body v priestore, ktoré reprezentujú kĺby ako napríklad krk, zápästie, rameno ale aj dlaň, či chodidlo. S ktorými môžu následne pracovať a vytvárať pohyby a gestá. Na obrázku číslo 2 je zobrazená zachytená osoba, ktorá je vykreslená pomocou skeletonu a depth stream. Na obrázku číslo 10 sú zobrazené body, ktoré dokáže kinect snímať pomocou nainštalovaných knižníc od microsoftu. programátor nemusí používať tieto body aby pracoval s kinectom. Môže si naprogramovať vlastné pomocou depth streamu. Štvrtý typ je audio stream, ktorý je získaný z mikrofónového poľa.

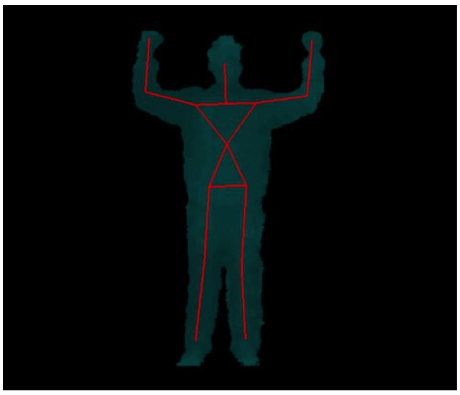
Kinect bol pôvodne vytvorený pre hernú konzolu Microsoft XBOX 360, ako ovládač pre hráčov. Je vytvorený spoločnosťou Microsoft. Pôvodný Kinect XBOX 360 sa od toho pre počítače nelíši. Jeho ukážka je na obrázku 1.

Kinect sa skladá z týchto komponentov:

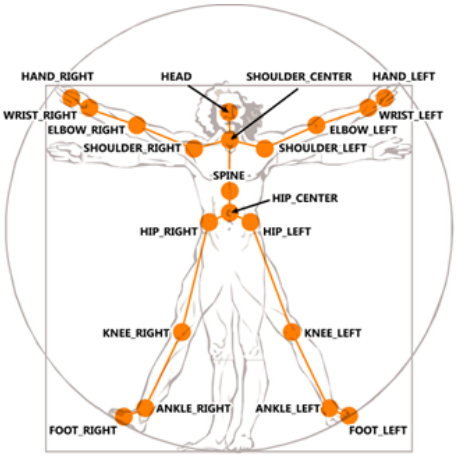
- Farebná RGB kamera s rozlíšením 1280x960 - slúži na snímanie užívateľov
- Hĺbkový(depth) senzor z rozlíšením 640x480.Slúži na snímanie hĺbkových dát s, ktorých získavame kostru snímanej osoby, poprípade rozpoznávanie tváre. A sa skladá z:
  - IR projektoru
  - IR Camera
- Pole štyroch mikrofónov na snímanie hlasu, kvôli rozpoznávaniu reči.
- LED indikátor - zapnutý / vypnutý
- Motorček - programovo ovládanie otáčanie kinectu

Jediná výnimka je káblová redukcia z výstupu kinect na USB a napájanie, ktorá dodáva potrebný príkon energie. Redukcia je zobrazená na obrázku 5. Zloženie kinectu je zobrazené na obrázku 4.

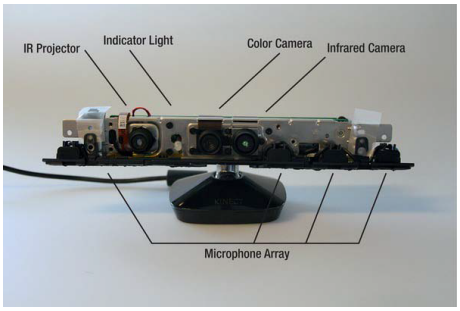
Obr. 1: Kinect senzor



Obr. 2: HĺbkovY stream s kostrou



Obr. 3: Kostra a všetky jej snímané body



Obr. 4: Kinect komponenty

2.1.1 Požiadavky na hardvér

Minimálne hardvérové nároky sú podobné výkonu konzoly XBOX 360.

- Procesor 2.66 GHz alebo rýchlejší
- Pameť 2GB - 4GB doporučené
- Grafická karta, ktorá podporuje DirectX9.0c

2.1.2 Požiadavky Kinect SDK na softvér

- Windows 7 x64 - x68
- Microsoft Visual Studio 2010 Express, alebo novšie štúdio
- Microsoft .Net framework 4.0, alebo novší
- Kinect SDK pre Windows

V prípade používania audio možností kinectu je treba nainštalovať - Rozpoznávanie reči od Microsoftu.

- Speech Platform API
- Speech Platform SDK
- Kinect for Windows Runtime Language Pack

2.1.3 Inštalácia Kinect pre Windows SDK

Pred inštaláciu SDK sa uistíme, že je senzor odpojený a Visual štúdio nainštalované a vypnuté. Ak máte nainštalované knižnice tretích strán doporučuje sa ich odinštalovať, kinect pre Windows SDK s nimi nebude spolupracovať. Iné ovládače môže spôsobovať nekonzistenciu a kinect nemusí pracovať správne.

Obr. 5: Kinect redukcia

Obr. 6: Správca zariadení

Následne si zo stránky Microsoftu treba stiahnuť inštalačný ".msi"balíček zodpovedajúci verzií Windows, teda x68 alebo x64. Po stiahnutí inštalovať podľa inštrukcií na obrazovke.

Po inštalácii sa môže pripojiť senzor, treba sa uistiť, že je pripojený aj dodatočný zdroj energie. Ak je kinect správne pripojený LED kontrolka bude svietiť na zeleno. Pre overenie, či je všetko nainštalované správne sa to dá overiť v: ovládacie panely / správca zariadení. Výsledok je zobrazený na obrázku 6.

2.2 Prečo vznikol kinect?

Microsft reagoval na vývoj trhu, kde sa už nachádzali zariadenie ako napríklad Nintendo Wii. Táto herná konzola umožňuje vývojárom a následne hráčom používať gestá pomocou rúk a bezdrôtových alebo drôtových ovládačov. Tieto ovládače snímajú kde sa

v priestore nachádzajú a vďaka tomu sa dajú vytvárať rôzne gestá, alebo ovládať prvky v hre. Napríklad hranie tenisu, kde v ruke držíme ovládač ako raketu.

Microsoft išiel technologicky ďalej a nahradil viacero zariadení, ktoré by museli byť pripojené na osobe iba jedným zariadením, ktoré sníma a rozlišuje osoby. Vďaka čomu Kinect dokáže využívať celú osobu a jej pohyby, alebo aj viacero osôb súčasne.

3 Možnosti SDK Kinect pre Windows

Programovať pre Kinect SDK je možné vo viacerých jazykoch. Microsoft podporuje tri jazyky C++, VB, C#. V tejto práci bude používaný jazyk C# spolu s technológiou WPF.

Na internete sa dajú nájsť aj knižnice pre prácou z jazykom JAVA alebo aj Matlab. Základné SDK je od výrobcu kinectu teda Microsoft, ale dá sa nájsť aj voľné SDK. Toto SDK obsahuje všetky potrebné knižnice k práci s kinectom.

3.1 Audio a video

Kinect obsahuje RGB kameru, ktorá dokáže nahrávať v rozlíšení 1280x960 pixelov. Pomocou tejto kamery môžeme zaznamenávať video a pomocou tried naprogramovaných v SDK môžeme do tohto streamu vkladať hĺbkové dáta, alebo aj kostru. Jednoducho môžeme zistiť, kde sa nachádza hocijaká časť tela. Napríklad potrebujeme portrét osoby, ktorá ovláda kinect, tak zistíme kde je hlava pomocou skeleton dát a urobíme si fotku. Veľmi často sa táto kamera využíva na zachytávanie fotiek pri rôznych úkonoch v hrách, ktoré sú zobrazené na konci hry. Táto kamera sa dá použiť aj ako obyčajná internetová kamera. Zároveň sa dá použiť aj ako mikrofón vďaka mikrofónovému poli.

Mikrofónové pole pozostáva z viacerých mikrofónov dokáže veľmi dobre zaznamenávať zvuky. Hlavná trieda pracujúca s Audio streamom je *KinectAudioSource*. Táto trieda má za účel posielať nespracovanú zvukovú stopu, alebo softvérovo upravenú. Zvuková stopa môže byť rôzne modifikovaná. Môže zvyšovať svoju kvalitu, taktiež môže potlačovať šum, odstraňovať ozveny. Dokonca môže zisťovať z, ktorej strany bolo na senzor hovorené a pomocou algoritmu ovládajúceho motorček sa môže natáčať za hlasom. Alebo nútiť mikrofón snímať zvuk primárne z vybranej strany, pre lepšiu kvalitu zvukovej stopy.

3.2 IR senzory

Pomocou IR senzorov je možne zaznamenávať hocijaké tvary, či už hľadáme kocku alebo osobu. Kinect SDK je primárne naprogramované na rozpoznávanie ľudí. SDK dokáže rozpoznať celé ľudské telo napríklad ruku, nohu, hlavu ale nedokáže rozpoznať prsty. Pomocou hĺbkových senzorov sme schopný naprogramovať rozpoznávanie hocijakých tvarov.

Rozpoznávanie prstov na ruke bolo naprogramované viacerými programátormi. Veľmi pekná video ukážka je spravená na adrese `http://www.youtube.com/watch?feature=player_embedded&v=tlLschoMhuE#!` Tento príklad je naprogramovaný študentom z MIT menom Garratt Gallagher, ktorý použil OpenKinect / libfreenect.

Každé nové SDK prináša rozpoznávanie niečoho nového Vo verzii SDK 1.5 prišlo rozpoznávanie tváre. Ukážka rozpoznávanie tváre v programe Kinect for Windows Developer Toolkit v 1.7.0. Aplikácia sa volá face Tracking 3D WPF. V tejto aplikácii nie je len rozpoznanie tváre, ale aj 3D zobrazenie pomocou WPF.

S poslednou verziou SDK 1.7.0 prišlo **Kinect Fusion** demonštračná aplikácia sa dá taktiež nájsť v programe Kinect for Windows Developer Toolkit v 1.7.0. Kinect fusion poskytuje 3D skenovanie objektu vytváranie 3D modelu pomocou senzoru kinect. Užívateľ môže vytvárať scénu a pracovať s objektmi, ktoré sú v nej.

Hardvérové požiadavky sú kvôli náročnosti veľmi vysoké. Procesor je doporučený viacjadrový 3GHz alebo silnejší. Grafická karta sa doporučuje aby podporovala DirectX 11 a mala aspoň 2GB internej pamäte. Kinect Fusion bol testovaný na Grafických kartách NVidia GeForce GTX680 a AMD Radeon HD 7850.

4 Gestá

Gesto v kinecte rozumieme hocijaký pohyb telom, ktorým sa dá niečo vyjadriť. Za gesto môžeme považovať mávanie, výskok, nahnutie tela do strán, ale aj omnoho jednoduchšie ako je nastavenie ruky na určitý bod obrazovky a zatlačenie, alebo pohyb ruky z ľavej do pravej strany. Tak isto existujú zložitejšie gestá, alebo kombinácia gest to závisí na programe a programátoroch, ktorý tieto gestá programujú.

Programovanie gest pre ovládanie nejakej hry je oveľa zložitejšie, ako ovládanie jednoduchého programu napríklad na prehrávanie videa, kde minimálne potrebujeme tri gesta a to pohyb ruky z ľavej do pravej strany pre posun obrazovky na ľavú stranu, alebo výber ďalšieho videa. Ďalej to iste pre posun do pravej strany, respektíve výber predchádzajúceho videa. A gesto pre spustenie a zastavenie videa je jednoduchý push teda zatlačenie. Týmito troma gestami sa dajú ovládať aj iné programy ako napríklad prehliadač obrázkov. A však v hre si programátor môže vymyslieť rôzne gestá, ktoré predtým postava vykonala po stlačení viacerých kláves. Napríklad výskok a následne rozpaženie, čo by sa dalo využiť v nejakej atletickej hre.

4.1 Typy gest

V tejto kapitole vám ukážem niekoľko gest, ktoré sú použité v mojich ukážkových aplikáciách. Ale aj gestá, ktoré nie sú použite v týchto aplikáciách.

4.1.1 Gesto zatlačiť

Toto gesto má nahradiť klik myši na počítačoch a dotyk na tabletoch. Existuje niekoľko variácií tohto gesta. Napríklad ruka sa podrží nad objektom a po ubehnutí zvoleného času sa toto vyhodnotí, ako klik. v tomto prípade sa osa Z teda hĺbková osa neberie v úvahu. Iná možnosť je, že sa os Z berie v úvahu a nad objektom sa vykoná zatlačenie. To znamená že sa rukou pohne smerom ku kinecte. Variácia tohto gesta je použitá v aplikácií KinectUI, odsek 7, kde funguje ako klasický klik. A v aplikácií Camera3D, odsek 6.2 sa používajú obidve ruky a toto gesto sa využíva pohyb dopredu. Microsoft dodal toto gesto v najnovšej verzii SDK v 1.7.0.

4.1.2 Gesto pohyb do ľavej a do pravej strany

Toto gesto slúži väčšinou na pohyb obrazovky alebo objektu do ľavej strany, alebo do pravej strany. Pre pohyb do ľavej sa používa pravá ruka. Pre gesto pohyb do pravej strany sa používa ľavá ruka. Gesto spočíva v pohybe natiahnutej ruky čo najviac na stranu a potom plynulým prechodom mierne za stred. Stred rozumieme aktuálnu pozíciu stredu ramien. Toto gesto sa využíva v aplikácií užívateľské rozhranie Kinect, odsek 7. Toto gesto sa používa vo veľa aplikáciách a vo viacerých variáciách. Jedná z variácii môže byť, že sa dá jednou rukou pohybovať obi dvoma smermi. Ďalšia z variácii môže byť použiť zároveň obi dve ruky na pohyb do jednej strany. Toto gesto by sa dalo využiť v

Obr. 7: Gesto pre pohyb hore a dolu

obrázkovej galérii, ako prechod na posledný obrázok, alebo v hre, ak by sa malo pohnúť niečím ťažkým.

4.1.3 Gesto pohyb hore a dole

Gesto slúžiace na rolovanie stránok. V zásade ide o gesto totožné s gestom **pohyb do ľavej a do pravej strany**, popísaného v odseku 4.1.2, akurát sa zaznamenávajú vertikálne súradnice teda súradnice Y. Ako príklad si môžeme predstaviť rolovanie internetovej stránky, alebo dlhého dokumentu. Pri zistení že sa jedná o toto gesto program posunie dokument hore alebo dole o nejakú časť. Toto gesto sa využíva v programe KinectUI, popísaného v odseku 7, ale nie ako rolovanie, ale ako tlačidlo späť. gesto je popísané v odseku Detekcia pomocou Algoritmu 4.2 a vo výpise číslo 1.Gesto je zobrazené na obrázku 7.

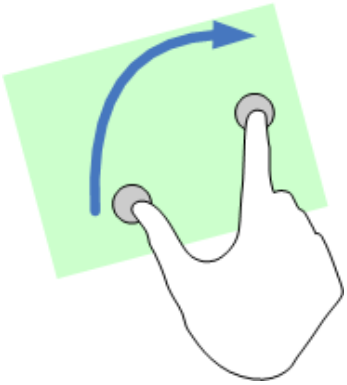
4.1.4 Gesto ťah

Toto gesto je kombinácia gesta push a pohybov do rôznych strán. Má nahradiť klik a potiahnutie myšou na PC a ťahanie na dotykových obrazovkách. Funguje na princípe gesta zatlačiť popísaného v odeku 4.1, keď sa toto gesto vykoná tak následne zoberiem virtuálny objekt a môžeme s ním pohybovať. Tiež sa môže využiť k rolovaniu stránok.

4.1.5 Gesto zväčšenie a zmenšenie

Na toto gesto sa využívajú obi dve ruky. Pre zväčšenie sa nastavia obi dve ruky čo najbližšie k sebe a plynulo sa od rozťahujú, až kým nie je zväčšenie postačujúce, alebo maximálne. Pre zmenšenie sa nastavia ruky čím viac od seba a postupne sa k sebe približujú. Alternatíva k využitiu rúk môžu byť prsty, ako je zobrazené na obrázku 8. Takáto implementácia je zložitejšia, pretože Kinect SDK nepodporuje rozpoznávanie prstov.

Obr. 8: Zväčšovanie pomocou prstov



Obr. 9: Otáčanie pomocou prstov

4.1.6 Otáčanie

Rotácia využíva obi dve ruky pričom ruky sú od seba v určitej vzdialenosti. Vzdialenosť nie je dôležitá. Gesto sa vykonáva súbežne obi dvoma rukami. A to pohybom jednej ruky do hora v kruhu smerom k stredu a druhej ruky dolu smerom k stredu. Ruky by mali ostať v rovnakej vzdialenosti, pretože toto gesto sa môže vo veľa aplikáciách spojiť s gestom zväčšenie a zmenšenie a tak využívať tieto gestá zväčšovanie, zmenšovanie a otáčanie, ako jedno gesto. Toto gesto sa dá taktiež použiť s použitím prstov. Gesto je zobrazené na obrazku 9.

4.2 Detekcia pomocou algoritmu

Programovanie gest relatívne jednoduché a môžu ho programovať programátori z relatívne malými skúsenosťami. Ale je veľmi obmedzené nedajú sa naprogramovať gesta ako napríklad hod, točenie. Tým že sa gestá programujú narastajú knižnice v ktorých sa nachádzajú a tým sa zvyšujú požiadavky na výkon.

Algoritmicky sa dajú rozpoznávať iba niektoré gestá, zväčša sú to gestá, ktoré nie sú zložité na vykonanie čo znamená že sa skladajú iba z jedného maximálne dvoch úko-

nov. Samozrejme dajú sa naprogramovať aj veľmi zložité gestá, ale ich programovanie je veľmi zložité a náročné. V prípade že je potrebné naprogramovať takéto gesto, musí sa rátať s tým že toto gesto by sa mohlo pretínať s iným gestom. Toto samozrejme neplatí iba pri zložitých gest.

Napríklad gesto pohyb rukou z hora dole, a zdola hore, ktoré môže slúžiť ako rolovanie v stránke z hora na dol a naopak. Programátorsky je veľmi obtiažné vytvoriť algoritmus, ktorý by rozpoznal, že sa nejedná o gesto ale že sa iba osoba snaží dostať ruku navrch obrazovky alebo strany aby mohla vykonať gesto rolovanie dolu, alebo nejaké iné gesto.

V nasledujúcom kóde 1 je napísaný kód pre detekciu gesta pohyb pravej ruky z hora dole. Tento kód slúži v mojom programe na prechod o úroveň späť teda funguje ako tlačidlo späť.

Ako vstupný parameter je pole double[] v ktorom sú zaznamenané aktuálne pozície ruky X, Y, Z. Na začiatku metódy sa zadefinujú hodnoty vrchného obdĺžnika odkiaľ sa zaznamenáva gesto. V tomto prípade je to pole double[] üp". Šírka je od 2/10 šírky strany po 8/10 šírky okna. Výška je od 0 teda od vrchu okna po 1/15 výšky strany. Ďalej je zadefinovaný obdĺžnik na spodku okna po ktorý sa zaznamenáva gesto. Jeho hodnoty sú:

- šírka - 2/10 až 8/10 šírky okna
- výška - 3/4 až koniec okna

Nasleduje podmienka na zistenie či je ruka v hornom obdĺžniku. Ak áno tak sa nastaví premenná _close na true, to zaistí že sa môže splniť ďalšia podmienka v, ktorej sa zisťuje či prešla prvá podmienka a či ruka stále klesá dolu. V _closeMoveHelpR[1] je uložená pozícia ruky pred aktuálnou. Ak ruka neklesá nastaví sa premenná _close na false a tým sa uzavrie možnosť o vykonanie gesta. V ďalšej podmienke sa zisťuje, či je ruka v druhom dolnom obdĺžniku, ak áno podmienka sa vyhodnotí, ako true a môže sa vykonať kód naviazaný na túto podmienku.

```
private bool Close_FromTop(double[] hand)
  {
      double[] up = new double[4];
      up[0] = (this._msc.ActualWidth / 10) * 2;
      up[2] = (this._msc.ActualWidth / 10) * 8;
      up[1] = 0;
      up[3] = this._msc.ActualHeight / 15;

      double[] down = new double[4];
      down[0] = (this._msc.ActualWidth / 10) * 2;
      down[2] = (this._msc.ActualWidth / 10) * 8;
      down[1] = this._msc.ActualHeight - this._msc.ActualHeight / 4;
      down[3] = this._msc.ActualHeight;

      if (hand[0] > up[0] && up[2] > hand[0] && hand[1] < up[3])
      {
```

```
            _close = true;
        }
        if (_close && _closeMoveHelpR[1] < hand[1])
        {
            if (hand[0] > down[0] && down[2] > hand[0] && hand[1] > down[1])
            {
                _close = false;
                return true;
            }
        }
        else
        {
            _close = false;
        }

        return false;
    }
```

Výpis 1: Detekcia gesta. Pohyb ruky z hora dolu

4.3 Detekcia pomocou neurónovej siete

Neurónové siete sa používajú už veľmi dlho, ale aj napriek tomu sú veľmi zložité na naprogramovanie a nastavenie. Málo ktorý programátor ich vie používať a aj tí, ktorí ich poznajú a používajú môžu mať problém s ich implementáciou v kinecte. A však rozpoznávanie gest pomocou neurónovej siete je časom veľmi presné a rýchle.

Ako príklad môžeme použiť výskok a podrep - drep. Gesto výskoku by sa jednoducho dalo definovať ako dvihnutie nôh od zeme, čo by nebolo ťažké naprogramovať nejakým jednoduchým algoritmom. Gesto drepu by sa dalo jednoducho naprogramovať priblížením bokov ku kolenám, alebo až pod kolená. No ak by mali byť tieto dve gestá v jednom programe súčasne rozpoznané nastal by problém. Ak by mala osoba vyskočiť vyššie prirodzene by sa prikrčila čo by sa dalo rozpoznať ako podrep, alebo ako podrep a vyskočenie čo nemusí byť požadovaní výsledok. Nikdy by tak nemuselo prísť k správnemu vyhodnoteniu.

Práve na takéto situáciu slúžia neurónové siete. Pri výskoku by dobre naprogramovaná neurónová sieť rozpoznala výskok. Napríklad na 90% by to bol výskok a podrep na 10%.

4.4 Detekcia pomocou príkladu

Detekcia gest pomocou príkladu je metóda štatistického porovnávanie nahratých dát. Tieto dáta sa získavajú nahrávaním veľkého množstva osôb, ktoré robia nejaké gesto. Kinect zaznamenáva hĺbkové body jednotlivých časti skeletonu a ukladá ich do databázy. Táto metóda rozpoznávania je časom najúčinnejšia, pretože žiadne gesto v podaní rôznych osôb nie je rovnaké.

Na príklad mávanie by sa dalo naprogramovať ako natiahnutím jednej ruky, otočenie v lakti do vertikálnej pozície a postupne mávanie z ľavej do pravej strany. Ale každý človek máva iným spôsobom. Tiež by sa toto gesto dalo naprogramovať ako mávanie celou rukou, alebo mávanie iba dlaňou . Samozrejme existuje kombinácia všetkých týchto typov dokopy. Takéto na prvý pohľad jednoduché gesto bo bolo na naprogramovanie pomocou algoritmov veľmi komplikované, preto sa požíva detekcia gest pomocou príkladu.

Časovo sa takáto detekcia môže zdať zdĺhavá, pretože treba veľa dát a ich zber zaberá čas, ale výsledky sú o veľa prirodzenejšie, ako keby bolo gesto naprogramované. Takéto metóda detekcie potrebuje veľkú pamäť a výkonný procesor pre porovnávanie gest v databáze.

5 Tutoriál

Následujúci tutoriál je detailny popis, ako vytvoriť funkčnú aplikaciu zobrazujúcu skeleton jednej osoby.

Pre tento tutoriál je potrebné mať nainštalované Visual študio 2012 a Kinect SDK 1.7 pre Windows, .net 4 alebo novší a samozrejme pripojený kinect. Vytvárať sa bude aplikácia v technológii WPF a za pomoci programovacieho jazyka C# .net 4.0.

1. vytvorenie WPF projektu vo Visual štúdiu s jazykom C#
2. pridanie referencií kinect nastavíme sa na MainWindow
3. pridanie using Microsoft.Kinect;
4. pridanie globálnej premennej KinectSensor sensor;
5. pridanie udalostí WindowLoaded a WindowClosed
6. v metóde Window_Closed treba nastaviť sensor.Stop(); aby sa vypol kinect po skončení programu. Viď kód číslo 2.

```
private void Window_Closed_1(object sender, EventArgs e)
  {
      sensor.Stop();
  }
```

Výpis 2: Window_Closed

7. v metode Window_Loaded sa zistí, či je kinect pripojený ak áno kinect treba zapnúť a priradiť mu eventHandler sensor_SkeletonFrameReady. Viď kód číslo 3

```
private void Window_Loaded_1(object sender, RoutedEventArgs e)
  {
      if (KinectSensor.KinectSensors.Count > 0)
      {
          sensor = KinectSensor.KinectSensors[0];
          sensor.Start();
          sensor.SkeletonStream.Enable();
          sensor.SkeletonFrameReady += new EventHandler<
              SkeletonFrameReadyEventArgs>(sensor_SkeletonFrameReady);
      }
      else
      {
          MessageBox.Show("Kinect nie je pripojeny");
          this.Close();
      }
  }
```

Výpis 3: Window_Loaded

8. vytvorenie metódy GetJointPoint - kód číslo 4, ktorá zisťuje, kde sa jednotlivé body nachádzajú vracia ich polohu prispôsobenú obrazovke. Jej vstupný parameter je joint, čo je jeden z bodov kinectom reprezentujúcich časti, kĺbov ľudského tela. Napríklad hlava : JointType.Head. Pri získavaní bodov X a Y sa môže použiť konštanta aby sa zväčšil dosah napríklad rúk. V tomto príklade je použitá horizontálna konštanta 1.2, a vertikálna 1.44.

```
private Point3D GetJointPoint(Joint joint)
{
    DepthImagePoint point = this.sensor.MapSkeletonPointToDepth(joint.Position,
    this.sensor.DepthStream.Format);
    point.X *= (int)(this.Grid1.ActualWidth * 1.2) / this.sensor.DepthStream.
        FrameWidth;
    point.Y *= (int)((this.Grid1.ActualHeight) * 1.44) / this.sensor.DepthStream.
        FrameHeight;

    return new Point3D(point.X, point.Y, point.Depth);
}
```

Výpis 4: GetJointPoint(Joint joint)

9. vytvorenie metódy CreateFigure - kód číslo 5, so vstupnými parametrami skeleton - snímaná kostra, osoba, brush - farba vytvorenej kostry, joints - pole jointov bodov, ktoré sa budú spolu prepájať. Výstupom je typ Polyline - body spojene čiarou.

```
private Polyline CreateFigure(Skeleton skeleton, Brush brush, JointType[] joints)
{
    Polyline figure = new Polyline();
    Grid.SetColumnSpan(figure, 5);
    Grid.SetRowSpan(figure, 5);
    Grid.SetZIndex(figure, -10);
    figure.StrokeThickness = 8;
    figure.Stroke = brush;
    for (int i = 0; i < joints.Length; i++)
    {
        Point3D point3D = GetJointPoint(skeleton.Joints[joints[i]]);
        figure.Points.Add(new Point(point3D.X, point3D.Y));
    }
    return figure;
}
```

Výpis 5: CreateFigure(Joint joint)

10. v ďalšom kroku treba nastaviť zopár pomocných premenných a jednu pomocnú metódu, ktorá zaisťuje aby bola zobrazovaná iba aktuálne snímaná kostra

    - skeleton - pole kostier(nasnímaných osôb)
    - skeletonActive - aktuálne snímaná kostra
    - activeSkeleton - boolean premenná zisťujúca je nejaká kostra aktuálne snímaná

Viď kód číslo 6.

```
Skeleton[] skeletons;
  byte skeletonActive = 0;
  bool activeSkeleton = false;

  private void SetActiveSkeleton(byte i)
  {
      skeletonActive = i;
      activeSkeleton = true;
  }
```

Výpis 6: pomocné premenné a metóda SetActiveSkeleton(byte i)

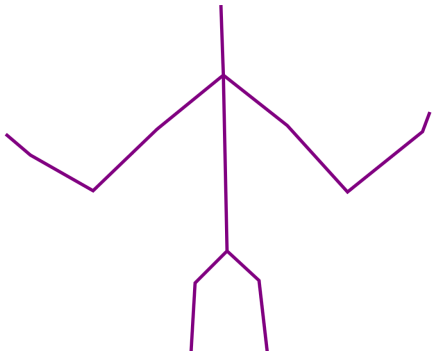
11. V nasledujúcom poslednom kroku treba doplniť EventHandler sensor_SkeletonFrameReady. Viď kód číslo 7. Táto metóda získava všetky rozpoznané kostry, ak sa nejaká nájde uloží sa jej hodnota, aby sa vykresľovala iba jedna kostra. Pomocou jointov sa dotazuje na metódu CreateFigure, ktorá vracia kostru a pridáva ju do gridu.

```
private void sensor_SkeletonFrameReady(object sender,
    SkeletonFrameReadyEventArgs e)
{
    using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
    {
        if (skeletonFrame != null)
        {
            if ((skeletons == null) || (skeletons.Length != skeletonFrame.
                SkeletonArrayLength))
            {

                skeletons = new Skeleton[skeletonFrame.SkeletonArrayLength];
            }
            skeletonFrame.CopySkeletonDataTo(skeletons);

            if (!activeSkeleton)
            {
                byte i = 0;
                foreach (Skeleton skeleton in skeletons)
                {
                    if (SkeletonTrackingState.Tracked == skeleton.TrackingState)
                    {
                        SetActiveSkeleton(i);
                    }
                    i++;
                }
            }
            else
            {
                if (skeletons[skeletonActive].TrackingState ==
                    SkeletonTrackingState.Tracked)
                {
                    if (skeletons[skeletonActive].Joints[JointType.HandLeft].
                        TrackingState == JointTrackingState.Tracked &&
```

```
                    skeletons[skeletonActive].Joints[JointType.HandRight].
                        TrackingState == JointTrackingState.Tracked &&
                    skeletons[skeletonActive].Joints[JointType.ShoulderCenter].
                        TrackingState == JointTrackingState.Tracked)
                {

                    Grid1.Children.Clear();

                    //ruky
                    JointType[] joints = new[] { JointType.Head, JointType.
                        ShoulderCenter, JointType.ShoulderLeft, JointType.
                        ElbowLeft, JointType.WristLeft, JointType.HandLeft };
                    Grid1.Children.Add(CreateFigure(skeletons[skeletonActive],
                        Brushes.Purple, joints));

                    joints = new[] { JointType.ShoulderCenter, JointType.
                        ShoulderRight, JointType.ElbowRight, JointType.
                        WristRight, JointType.HandRight };
                    Grid1.Children.Add(CreateFigure(skeletons[skeletonActive],
                        Brushes.Purple, joints));
                    //nohy
                    joints = new[] { JointType.ShoulderCenter, JointType.
                        HipCenter, JointType.HipLeft, JointType.KneeLeft,
                        JointType.FootLeft };
                    Grid1.Children.Add(CreateFigure(skeletons[skeletonActive],
                        Brushes.Purple, joints));

                    joints = new[] { JointType.HipCenter, JointType.HipRight,
                        JointType.HipRight, JointType.FootRight };
                    Grid1.Children.Add(CreateFigure(skeletons[skeletonActive],
                        Brushes.Purple, joints));
                }

            }
            else
            {
                activeSkeleton = false;
            }
        }
    }
}
```

Výpis 7: EventHandler sensor_SkeletonFrameReady

Po skompilovaní a spustení programu vznikne kostra, ktorá sa pohybuje podľa snímanej osoby. Výsledok ukážkového programu je na obrázku 10.



Obr. 10: Ukážka programu

6 Aplikacia 3D Camera

Aplikácia "3D Camera"je zameraná na využitie kinectu v mapách. Zámer bol demonštrovať pohyby v 3D mapách bez pomoci iných zariadení, iba s pomocou kinectu. Táto aplikácia je napísaná technológiou WPF a v jazyku C#. Použité vývojove prostredie je Microsoft Visual štúdio 2012. Mapa bola vygenerovaná pomocou programu Blender 2.66a a jazyku Python, a následne prevedená na XAML za pomoci skriptu "Blender 2.5 WPF XAML Exporter"vytvoreného Charlesom Hetierom. Tento skript sa dá stiahnuť na stránkach `http://charly-studio.com/blog/blender-2-5-wpf-xaml-exporter/`. Minimalne požiadavky sú také ako má Kinect SDK uvedené v sekcii 2.1.1. Požiadavky sa zvyšujú v závislosti na veľkosti mapy a jej kvality.

6.1 Architektura

Základom aplikácie je trieda MainWindow.xaml a jej kód na pozadí MainWindow.xaml.cs. Táto trieda je spúšťacia. Jej základom je hlavný grid, v ktorom sa zobrazuje mapa. Do tohto gridu je pridaných ďalších osem gridov ľavý, pravý, vrchný - (ľavý, stredný, pravý) a spodný - (ľavý, stredny, pravý). Tieto gridy slúžia na ovládanie mapy.

Kostra aplikácie je rovnaká ako v tutoriále 5. Trieda ma malé úpravy aby mohla spustiť UC triedu a nastavovať aktuálnu pozíciu kamery pomocou gest.

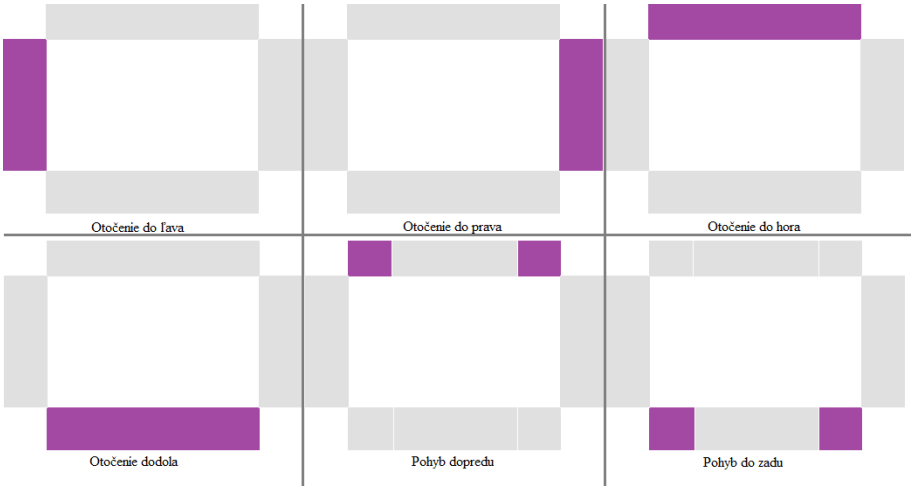
6.2 Gestá

Gestá v tejto aplikácii sú jednoduché, aby sa urýchlilo ovládanie. Dotykom na ľavý grid sa mapa otočí v ľavo, na pravý grid v pravo, na vrchný stredný hore, na spodný stredný dole. Ak sa dotkneme oboma rukami vrchného ľavého a vrchného pravého posunieme sa dopredu v smere, kde sme otočený. V prípade cúvania sa treba dotknúť spodného ľavého a spodného pravého gridu. Tieto gesta sú zachytené na obrázku číslo 11.

Pohyb dopredu sa dá vykonať aj gestom push. Toto gesto funguje na princípe zatlačenia obidvoch rúk dopredu . Algoritmus sníma polohu dlane a centra medzi ramenami. Následne tieto pozície porovnáva a zisťuje či je väčší ako konštanta, ktorá je v tomto prípade nastavená na 550. Táto konštanta bola získaná metódou skúšania a reprezentuje približnú vzdialenosť dlane a stredu ramien. Ak je táto pozícia väčšia ako 550 vykoná sa príslušná metóda. Táto metóda je zobrazená v kóde číslo 8.

```
private readonly double _pushDistance = 550;

private bool Push(Point3D hand, Point3D SH_CENTER)
  {
      if ((SH_CENTER.Z − _pushDistance > hand.Z))
      {
          return true;
      }
      return false;
  }
```

Výpis 8: Metóda pre pohyb dopredu



Obr. 11: Gesta v programe 3D Camera

Ostatné gestá funguju na princípe dotyku. Ak sa bod ocitne na nejakom gride vyvolá to akciu, ktorá sa ma vykonať. Algoritmus pre získavanie bodu v gride je ukázaný v kóde číslo 9. Vstupnými parametrami sú point3D čo je bod reprezentovaný súradnicami X,Y,Z. tento bod je prvá alebo ľavá dlaň. Druhým parametrom je grid. Zadáva sa grid ktorý sa porovnáva. Výstupom je boolean. Základom sú dve podmienky prvá z nich porovnáva či je bod medzi súradnicami gridu X a druhá či je medzi súradnicami gridu Y . Ak áno vyfarbí zadaný grid a vráti true. V takomto prípade sa vykoná akcia, ktorá je určená.

```
private bool IsPointInGrid(Point3D point3D, Grid grid)
  {
      BrushConverter br= new BrushConverter();
      Point  point  = grid.PointToScreen(new Point());
      if  (point.X < point3D.X && point.X + grid.ActualWidth > point3D.X)
      {
          if  (point.Y < point3D.Y && point.Y + grid.ActualHeight > point3D.Y)
          {
              grid.Background = (Brush)br.ConvertFrom("#335169A6");

              return true;
          }
      }
      grid.Background = (Brush)br.ConvertFrom("#33CBCBCB");

      return false;
  }
```

Výpis 9: Metóda zisťovanie bodu v gride

7 Aplikácia uživateľské rozhranie Kinect

Táto aplikácia demonštruje jednoduché ovládanie videí, obrázkovej galérie, emailu a RSS čítačky pomocou gest, ktoré sú vytvorene mnou . Tieto gestá sú napísané pomocou algoritmov ako je uvedené v sekcii 4.2. Aplikácia je napísaná v jazyku C# a technológiou WPF. Ako vývojové prostredie je použité vývojové prostredie Microsoft Visual štúdio 2012. Minimálne a doporučené požiadavky sú také ako má Kinect SDK uvedené v sekcii 2.1.1.

7.1 Architektura

Základom tejto aplikácie je trieda MainScreen.xaml a jej kód v pozadí MainScreen.xaml.cs. Bližší popis triedy je v odseku 7.2. Táto trieda zobrazuje všetky UC triedy a prideľuje im rozpoznané gesta.

Pri spustení programu sa spustí trieda MainScreen, ktorá načíta všetky potrebné UC triedy pre beh programu. Potom zapni senzor kinect a zobrazí UC triedu ChoosUser. Po tom, ako kinect nájde skeleton sa začne detekcia gest pomocou triedy ActionCtrl popísanej v odseku číslo 7.3.2. Ak je gesto rozpoznané, trieda vráti názov vykonaného gesta. Trieda MainScreen zistí aká UC trieda je zobrazená a pošle jej toto gesto, kde sa vykoná potrebná operácia.

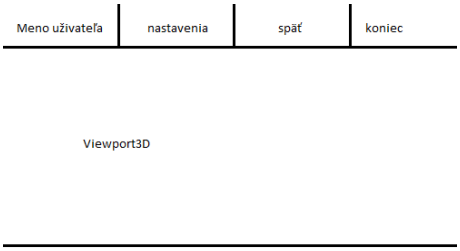
Program obsahuje šesť UserControl tried, ktoré sa môžu zobrazovať.

- ChooseUserCtrl - trieda pre zobrazenie a výber užívateľa, táto UC trieda sa používa ako prvá, hneď po zapnutí programu
- MainScreenCtrl - táto trieda obsahuje obrázky reprezentujúce odkazy na ostatné UserControl triedy, ktoré sa majú zobraziť. Zobrazí sa po výbere užívateľa
- PicturesCtrl - trieda pre zobrazenie obrázkovej galérie
- MessagesCtrl - trieda pre zobrazenie emailov, v tejto ukážkovej aplikácií je použitý Gmail.
- RssCtrl - trieda zobrazujúca RSS správy
- YoutubeCtrl - trieda zobrazujúca youtube odbery

Všetky triedy sú popísané v odseku číslo 7.4 a v 7.3.

7.2 MainScreen

Základ dizajnovej časti je grid, ktorý je rozdelené na ďalšie tri gridy, ako je zobrazené na obrázku číslo 12.V prvom gride sú tri tlačidla a jeden label. Do labelu sa ukladá menu aktuálneho užívateľa, prvé tlačidlo slúži pre nastavania používateľov, ďalší slúži na vrátenie späť ak by sa kinect zasekol, alebo by sa bolo nutné vrátiť bez pohybu využitia



Obr. 12: Ukážka rozloženia aplikácie kinect

kinectu. A koniec na ukončenie celého programu. Stredný grid obsahuje jedinú komponentu a to Viewport3D a ten pomocou kódu na pozadí zobrazuje UserControl triedy. Tieto triedy slúžia na zobrazovanie všetkého obsahu, ktorý sa má zobraziť.

Táto trieda je základom celej aplikácie z veľkej časti je podobná triede Window z tutoriálu popísaného v odseku 5. Totožná je napríklad metóda pre zistenie, či je kinect pripojený, a zapnutie senzorov, ako je v kóde číslo 3. Ostatné metódy sú s malými úpravami veľmi podobné.

Je o veľa rozsiahlejšia ako tá v tutoriále, ale to jadro používania kinectu je rovnaké. Pri inicializácii sa načítajú všetky potrebné UC triedy a iné triedy, ktoré sú potrebné pre beh programu. Táto trieda obsahuje mnou vymyslenú metódu **SetElementPosition** popísanú v kóde číslo 10, pre zobrazovanie bodov kinectu dalo by sa povedať že je to ekvivalent triedy GetJointPoint z kódu číslo 4 z tutoriálu. Metóda GetJointPoint využíva Microsoftom napísanú triedu MapSkeletonPointToDepth, ktorá zisťuje aktuálnu pozíciu jointov a zobrazuje ich pozíciu na obrazovke.

Metóda SetElementPosition robí presne to iste ale nepožíva žiadnu z knižníc od Microsoftu. Jediné čo používa je joint.Position, ktorá vráti pozíciu bodu získaného z kinectu. Tato súradnica je v desatinnom formáte menšom ako 0. Napríklad 0,155784216. Preto sa musí prenásobiť konštantou. V prípade, že by sa jednalo o X súradnicu a chceli by sme dosiahnuť aby sa bod nepohyboval mimo obrazovky tak by sme ho vynásobili šírkou obrazovky a pripočítali stred obrazovky, ak by sme stred nepripočítali tak by sa kostra, alebo bod pohyboval na ľavej strane obrazovky. To iste musíme vykonať aj zo súradnicou Y a Z. V prípade Y musíme získaný bod prenásobiť -1, aby sme sa dostali na plusové hodnoty s, ktorými sa lepšie pracuje a WPF zobrazuje svoje prvky na kladnej osi.

Zároveň táto metóda určuje polohu zadanému elementu zo vstupného parametru. Tento parameter nie je povinný.

```
readonly int _calibrateX = 2000, _calibrateY = 1600, _calibrateZ = 1200;

private double[] SetElementPosition(FrameworkElement elem, Joint joint)
  {
      try
      {
          _startX = this.ActualWidth / 2;
          _startY = this.ActualHeight / 2;
```

```
            double[] pos = new double[3];

            pos[0] = Math.Truncate((joint.Position.X * _calibrateX)) + _startX;
            pos[1] = -Math.Truncate((joint.Position.Y * _calibrateY)) - _startY;
            pos[2] = Math.Truncate(joint.Position.Z * _calibrateZ);

            if (elem != null)
            {
                elem.Margin = new Thickness(pos[0], pos[1], 0, 0);

                Point p = elem.PointToScreen(new Point());
                pos[0] = p.X;
                pos[1] = p.Y;
            }
            return pos;
        }
        catch (Exception)
        {
        }
        return null;
    }
```

Výpis 10: Metóda SetElementPosition

Po vykonaní gesta, sa toto gesto musí vykonať v správnej UC triede na toto slúžia 2 metódy, a to OpenCtrl a MoveControl. **OpenCtrl** je metóda, ktorá nastavuje stringovú premennú openControl na aktuálnu hodnotu UC triedy, ktorá je aktuálne zobrazená. Podľa tejto premennej sa zisťuje, ktorá UserControl trieda je momentálne nastavená. Taktiež vymaže pôvodný grid a nastaví ho na požadovaný grid.

Funkcia metódy **MoveControl** zaistiť že sa vykonané gesto dostane do správnej UC triedy. Metóda obsahuje switch s parametrom openControl, ktorý sa nastavuje v metóde OpenCtrl. Táto premenná má v základe nastavenú hodnotu na ChooseUserCtrl, čo znamená že sa ako prvá obrazovka zobrazí výber užívateľov.

7.3 Popis použitých pomocných tried

7.3.1 FindObject

Trieda, ktorá vyhľadáva všetky vizuálne objekty jedného typu v objekte, ktorý dedí z DependencyObject. Napríklad hľadá všetky Label objekty v triede MainScreen. Trieda FindObject obsahuje list závislých objektov, a metódu FindInVisualTreeDown, ktorá je zobrazená v kóde číslo 11. Metóda obsahuje dva vstupné parametre DependencyObject pre objekty, v ktorých sa bude vyhľadávať a Type pre typ objektu, ktorý sa bude vyhľadávať. Táto metóda je rekurzívna. Inspiroval som sa stránkou [2] presnejšie `http://stackoverflow.com/questions/974598/find-all-controls-in-wpf-window-by-type`.

```
        public List<DependencyObject> objDepList = new List<DependencyObject>();

        public DependencyObject FindInVisualTreeDown(DependencyObject obj, Type type)
          {
```

```
        if (obj != null)
        {
            for (int i = 0; i < VisualTreeHelper.GetChildrenCount(obj); i++)
            {
                DependencyObject child = VisualTreeHelper.GetChild(obj, i);
                if (child.GetType() == type)
                {
                    objDepList.Add(child);
                }

                DependencyObject childReturn = FindInVisualTreeDown(child, type);
                if (childReturn != null)
                {
                    return childReturn;
                }
            }
        }

        return null;
    }
```

Výpis 11: Trieda FindObject.cs

7.3.2 ActionCtrl

Táto trieda je základom celého programu, čo sa týka využívania kinectu a gest. Sú tu naprogramované metódy všetkých gest, ktoré sú použité v programe. Konštruktor tejto triedy obsahuje jediný parameter a to MainScreen. Ďalej ma dve public premenne, ktorými sú double polia leftHand a rightHand. Public sú preto aby si mohla hocijaká iná trieda zistiť aká je aktuálna pozícia rúk ak, sa práve vyvolá gesto OPEN _RIGHT alebo OPEN_LEFT. Tieto dve gesta fungujú, ako push na dotykový obrazovkách.

Veľmi dôležitá metóda je **SetAction**, kód číslo 12, táto metóda zisťuje, či sa jedná o gesto a zaisťuje, aby sa nevykonávali dve rôzne gesta naraz. Volá ostatné metódy, v ktorých sú algoritmy na získavanie a overovanie jednotlivých gest. Každá s týchto metód ma výstupný parameter boolean a vstupné parametre sú double polia reprezentujúce bod v trojrozmernej X, Y, Z polohe niektorého bodu na kostre osoby. Napríklad dlaň.

```
    public void SetAction(double[] left, double[] right, double[] SH_CENTER)
    {
        leftHand = left;
        rightHand = right;
        if (!_moveToRight && !_moveToLeft && !_close)
        {
            if (Open_PushRightHand(right, SH_CENTER))
            {
                this._msc.MoveControl("OPEN_RIGHT");
                _rHelpOpen = null;
            }
```

```
            if (Open_PushLeftHand(left, SH_CENTER))
            {
                this._msc.MoveControl("OPEN_LEFT");
                _lHelpOpen = null;
            }
        }
        if (!_moveToRight && !_open && !_close)
        {
            if (ToLeft_RHand(right))
            {
                this._msc.MoveControl("TO_LEFT");
            }
        }
        if (!_moveToLeft && !_open && !_close)
        {
            if (ToRight_LHand(left))
            {
                this._msc.MoveControl("TO_RIGHT");
            }
        }
        if (!_moveToRight && !_moveToLeft && !_open)
        {
            if (Close_FromTop(right))
            {
                this._msc.MoveControl("CLOSE");
            }
        }
        _moveToLHelp = right;
        _moveToRHelp = left;
        _closeMoveHelpR = right;
    }
```

Výpis 12: Metóda SetAction

Open_PushRightHand kód 13 je metóda pre gesto push pravou rukou. Výstupný parameter je boolean, vstupný parametre je pole double hand, ktorý reprezentuje kinect joint pravej dlane, a pole double SH_CENTER reprezentujúci kinect joint stredu ramien. Metóda zisťuje či je ruka v pozícii od stredu ramien viac ako 550. Ak áno nastaví sa pomocná premenná _rHelpOpen. V ďalšom bode sa zistí či je táto premenná nastavená, ak áno zisťuje sa či sa ruka drží v štvorci o rozmeroch x,y = 60 x 60. JE to preto aby sa naozaj vykonal push iba na zvolený objekt na obrazovke. Nakoniec sa zistí či ruka prekonala hĺbkovú vzdialenosť 60 v zadanom štvorci od pôvodnej polohy, kde sa začalo vykonávať gesto. Toto gesto je veľmi podobné gestu programu 3DCamera, ktoré je zobrazené v kóde číslo 8. Toto isté gesto sa dá vykonať aj ľavou rukou pre, ktorú je napísaný taký istý kód s malými úpravami. namiesto _rHelpOpen je premenná _lHelpOpen. Je to kvôli tomu aby sa dá dalo rozlíšiť, o ktoré gesto sa jedna.

```
    private bool Open_PushRightHand(double[] hand, double[] SH_CENTER)
    {
        if ((SH_CENTER[2] − 550 < hand[2]))
        {
```

```
            _rHelpOpen = null;
        }
        //  z − suradnice
        if (_rHelpOpen == null && SH_CENTER[2] − 550 > hand[2])
        {
            _rHelpOpen = hand;
        }
        if (_rHelpOpen != null)
        {   //  x−suradnice                                                    y−
            suradnice
            if (hand[0] < _rHelpOpen[0] + 60 && hand[0] > _rHelpOpen[0] − 60 && hand[1] <
                _rHelpOpen[1] + 60 && hand[1] > _rHelpOpen[1] − 60)
            {
                //z− suradnice
                if (hand[2] < _rHelpOpen[2] − 60)
                {
                    return true;
                }
            }
            else
            {
                _rHelpOpen = null;
                // _msc.KillOpenAnim();
            }

        }
        return false;
    }
```

Výpis 13: Metóda Open_PushRightHand

ToLeft_RHand je metóda, ktorá zisťuje či sa použilo gesto pohyb ruky z pravej do ľavej strany. Výstupný parameter je boolean, vstupný parametre je pole double[] hand, ktorý reprezentuje kinect joint pravej dlane. Metóda zistí pozíciu pravej strany obrazovky a stredu obrazovky. Gesto sa vykoná ak dlaň prekoná vzdialenosť od bodu zadaného v premennej pos po bod zadaný v bode middle a medzitým sa nesmie vrátiť späť. Metóda je zobrazená v kóde číslo 14. Takáto istá metóda existuje pre ľavú ruku pre pohyb z ľavej do pravej strany. Tieto metódy by sa dali použiť aj naopak, a to tak že pravá ruka by vykonávala gesto s ľavej do pravej strany a ľavá ruka by vykonávala gesto pohyb z pravej do ľavej strany, ale nie je to tak použité kvôli kinectu. Kinect nezvláda dobre ak sa má ľavá ruka dostať cez stred tela na druhú stranu a naopak. Vtedy stráca spojenie s niektorými bodmi, respektíve sa mu spájajú body.

```
    private bool ToLeft_RHand(double[] hand)
    {
        double[] pos = new double[4];
        pos[0] = this._msc.ActualWidth − (this._msc.ActualWidth / 10);
        pos[2] = this._msc.ActualWidth;
        pos[1] = this._msc.ActualHeight / 6;
        pos[3] = this._msc.ActualHeight − (this._msc.ActualHeight / 6);
```

```
        Point middle = new Point();
        middle.X = this._msc.ActualWidth / 2;
        middle.Y = this._msc.ActualHeight / 2;

        if (hand[0] > pos[0] && hand[1] > pos[1] && pos[3] > hand[1])
        {
            _moveToLeft = true;
        }
        if (_moveToLeft && _moveToLHelp[0] >= hand[0])
        {
            if (hand[0] > middle.X - 50 && middle.X + 50 > hand[0])
            {
                _moveToLeft = false;
                return true;
            }
        }
        else
        {
            _moveToLeft = false;
        }

        return false;
    }
```

Výpis 14: Metóda ToLeft_RHand

V tejto triede sa nachádzajú aj iné metódy pre zisťovanie gest a to Open_PushLeftHand, ktorá je s malými úpravami popísaná v kóde 13, ďalej metóda ToRight_LHand je s malými úpravami popísaná v kóde číslo 14, a metóda Close_FromTop, ktorá je popísaná v kóde číslo 1.

7.3.3 FolderList

Trieda obsahuje dve metódy GetDirectories a GetFiles. **GetDirectories** vyhľadáva podadresáre v zložke. Má jeden vstupný parameter string location, je to cesta, v ktorej sa majú vyhľadať podzložky. Výstupom je list DirectoryInfo. **GetFiles** vyhľadáva jpg, png a bmp súbory v zložke.

7.3.4 Images

Trieda vyhľadáva v dokumente Settings.xml všetky cesty k obrázkom od jedného užívateľa. Časť kódu je zobrazená v kóde číslo 15. Táto trieda sa používa iba v UserControl PicturesCtrl. Je to jej pomocná trieda.

```
        XmlNode noda = doc.SelectSingleNode("//Users/User[@ID=" + id + "]/Pictures");
            if (noda != null)
            {
                foreach (XmlNode a in noda)
```

```
        {
            list .Add(a.InnerText);
        }
        return list ;
    }
```

Výpis 15: Vyhľadávací algoritmus v Xml dokumente pre obrázky

7.3.5 MainPanel

Vytvára vzhľad pre zadaný objekt. Obsahuje dve triedy AddImagePanel a AddImageGalleryPanel. Tieto triedy obsahujú definíciu vzhľadu vstupného elementu. Výstupom je FrameworkElement, ktorý je popísaný v kapitole7.3.7.

7.3.6 Create3DImage

Táto trieda sa taktiež používa iba v UserControl PicturesCtrl **??**. Pomocou nej sa vytvára 3D vzhľad obrázku v galérii. Ako vstupné parametre má double offsetX, double offsetY, double offsetZ, FileInfo fi, double angle. Offsety sú pozícia, kde sa má obrázok zobraziť, angle - uhol v akom sa má zobraziť a FileInfo je vstupný súbor v tomto prípade obrázok. Využíva triedu popísanú v odesku 7.3.5 a nej metódu AddImageGalleryPanel, kde je nastavený vzhľad panelu, ktorý bude zobrazený. Výstupný parameter je Viewport2DVisual3D čo je objekt pretvarujúci 2D objekt na 3D.

7.3.7 MainScreen3D

Je takmer totožná s triedou Create3DImage, zmenou je že nepoužíva triedu MainPanel 7.3.5, čo znamená že vzhľad už je definovaný pri vstupe. Ma inak nastavenú os rotácie a veľkosť. Namiesto vstupného parametru FileInfo má vstupný parameter FrameworkElement. FrameworkElement reprezentuje WPF objektu definované v UIElement.

7.3.8 RssReader

Pomocná trieda pre UserControl RssCtrl. V tejto triede sa vykonáva sťahovanie a parsovanie RSS správ. Získavanie listu práv z xml dokumentu Settings.xml pre daného používateľa.

7.3.9 YoutubeClass

Pomocná trieda pre UserControl YoutubeCtrl. Trieda slúži na získanie youtube odberov. Overovanie či tieto kanály/odbery existuju. Získavanie ich kanálov. A vyhľadávanie youtube dlaždíc na obrazovke pomocou metódy FindTiles. Táto trieda používa triedu FindObject ktorá je zobrazená v 11 k tomu, aby našla všetky typy "Tileöbjektov. Následne

metóda získa ich polohu a vráti ich v liste. Slúži to k tomu, aby sa dala pomocou gesta push jednoducho nájsť príslušná dlaždica.

7.4 Popis použitých UserControl tried

Každá trieda, ktorá ma označenie xxxxCtrl.xaml sa zobrazuje v triede MainScreen.xaml. Všetky triedy obsahujú metódu DoAction. Kód číslo 16. Táto metóda je základom pre gestá, ktoré sú v programe použité. Má jeden vstupný parameter string. Táto metóda sa volá vždy s MainScreen.xaml.cs. je to veľmi jednoduchá metoda obsahujúca switch a v ňom preddefinované všetky možné gestá, ktoré môžu prísť do danej triedy a metódy, ktoré sa majú vykonať.

```
public void DoAction(string side)
  {
      switch (side)
      {
          case "OPEN_LEFT":
              {
                  SetUser(side);
              } break;
          case "OPEN_RIGHT":
              {
                  SetUser(side);
              } break;
          case "CLOSE":
              {
                  this._msc.EndProgram();
              } break;
          case "TO_LEFT":
              {
                  MoveToLeft();
              } break;
          case "TO_RIGHT":
              {
                  MoveToRight();
              } break;
          default: break;
      }
  }
```

Výpis 16: Metóda DoAction v UC ChooseUser

Každá takáto trieda má nadefinovaný vzhľad ako ma vyzerať, nejakú metódu ktorá vyhľadáva objekty pre gesto push ak toto gesto potrebuje. Animácie pre pohyby do strán, ak je potrebné. Klasická animácia pre pohyb z ľavej do pravej strany je zobrazená v kóde číslo 17. Všetky animované komponenty sú zobrazované 3D, preto stačí robiť naozaj jednoduché animácie pohybom kamery. Najskôr sa vyhľadá aktuálna pozícia kamery. Tieto parametre sa predaju animácii, ktorá sa vykoná.

```
if ( listPosition  > 0)
    {
        PerspectiveCamera p1 = mainViewPort.Camera as PerspectiveCamera;
        double x = p1.Position.X;
        double z = p1.Position.Z;
        double y = p1.Position.Y;

        Point3DAnimation pa = new Point3DAnimation();
        pa.BeginTime = new TimeSpan(0, 0, 0);
        pa.Duration = new TimeSpan(0, 0, 0, 0, 300);
        pa.From = new Point3D(x, y, z);
        pa.To = new Point3D(x − 2.4, y, z);
        mainViewPort.Camera.BeginAnimation(PerspectiveCamera.PositionProperty, pa);

         listPosition −−;
    }
```

Výpis 17: Animácia z ľavej do pravej strany

7.4.1 Príklad UC triedy ChooseUserCtrl

Dizajnový základ je grid a v ňom Viewport3D s definouvanou kamerou a svetlom, zobrazený obsah sa definuje v .cs triede.

Popis použitých metód

Konštruktor má jeden parameter MainScreen, kvôli previazaniu UC s programom. V metóde sa inicializujú komponenty a zavolá sa metóda FillViewPort.

GetUser

FillViewPort vyplní viewport všetkými užívateľmi, ktorý sú zaregistrovaný v programe. Najskôr získa všetkých užívateľov za pomoci triedy GetUser. Potom vytvorí prázdny list objektov Viewport2DVisual3D. Do tohto listu sa postupne vkladajú v cykle "for"vytvorené 3D objekty typu MainScreen3D. Keď je list naplnený tak sa pomocou cyklu "foreach"naplní mainVievPort, čo je komponent Viwport3D. Výsledok je zobrazený na obrázku 13.

```
List<TilesObj> users = GetUser();
    _tiles  = users;
    MainScreen3D ms3 = new MainScreen3D();

    List<Viewport2DVisual3D> list = new List<Viewport2DVisual3D>();
...
     list .Add(ms3.Show3dList(x, y, 0, new Tile(users[pos].name, users[pos].path), 0));
     ...
    foreach (Viewport2DVisual3D a in list)
    {
        mainViewPort.Children.Add(a);
    }
```

Výpis 18: FillViewPort()

Obr. 13: Ukážka programu. Zobrazená trieda: ChooseUserCtrl

Ovládanie tejto UC triedy je zaistené triedou **DoAction**, ktorá je popísana v kóde číslo 16. Táto metóda môže vykonať štyri metódy. A to sú metódy pre výber uživateľa **SetUser**, alebo pohyb obrazovky do ľavej alebo do pravej strany: **MoveToLeft**, **MoveToRight**. Alebo ukončenie programu, kde sa odkazuje na triedu MainScreen a metódu EndProgram.

SetUser reaguje na gesto push a teda musí zistiť či sa gesto vykonalo, keď bola dlaň nastavená nad nejakým objektom, alebo bola mimo. Najskôr sa zavolá metóda **FindTiles**, popísaná v kóde číslo 20, ktorá nájde všetky požadované objekty. Táto metóda sa musí volať zakaždým, pretože poloha objektov a môže meniť. potom sa zavolá metóda **ComparePositions**, popísaná v kóde číslo 19, ktorá overí či je joint na nejakom z nájdených objektov. Ak sa táto zhoda potvrdí nastaví sa vybraný používateľ a zobrazí sa nová UC trieda MainScreenCtrl.

```
private TilesObj ComparePositions(double[] pos)
  {

      foreach (TilesObj u in  _tiles )
      {
          if  (u.points[0]  < pos[0]  && pos[0] < u.points[2]  && u.points[1]  < pos[1]  && pos[1]
              < u.points[3])
          {
              return u;
          }
      }
      return null;
  }
```

Výpis 19: ComparePositions()

FindTiles používa na vyhľadávanie metódu FindInVisualTreeDown. Kód číslo 11. Táto metóda vráti všetky požadované objekty. Metóda naplní list _tiles, ktorý slúži na

porovnávanie v metóde ComparePositions. Kód číslo19.

```
private void FindTiles(DependencyObject obj)
  {
      FindObject fo = new FindObject();

      fo.FindInVisualTreeDown(obj, typeof(Tile));
      if (fo.objDepList != null)
      {
          List<DependencyObject> objList = fo.objDepList;
          int i = 0;
          foreach (Tile a in objList)
          {
              try
              {
                  Grid grid = (Grid)a.FindName("grid");
                  Image im = (Image)a.FindName("img");
                  TextBlock t = (TextBlock)a.FindName("lbl");

                  double with = grid.ActualWidth;
                  double heith = grid.ActualHeight;
                  Point point = grid.PointToScreen(new Point());
                  double[] pointsXY = new double[4];
                  pointsXY[0] = point.X;
                  pointsXY[1] = point.Y;
                  pointsXY[2] = point.X + with;
                  pointsXY[3] = point.Y + heith;

                  _tiles[i].points = pointsXY;
                  i++;
              }
              catch (Exception)
              {
              }
          }
      }
  }
```

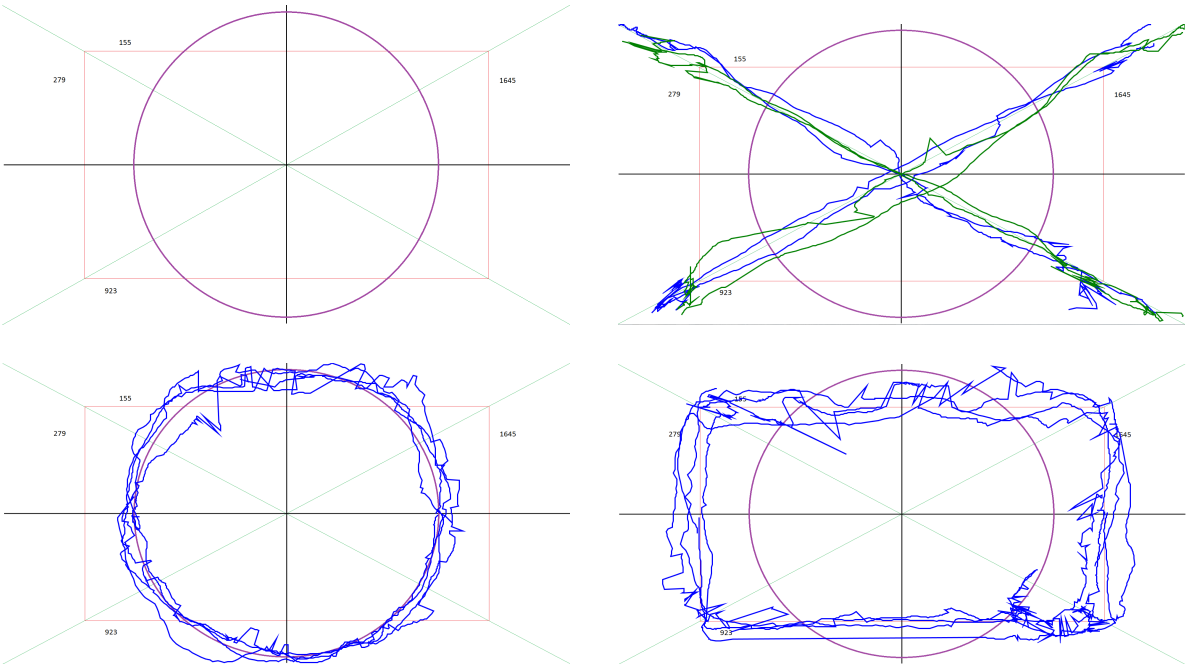
Výpis 20: FindTiles()

Metódy **MoveToLeft** a **MoveToRight** slúžia na posúvanie obrazovky pomocou animácie v prípade že sa všetci užívatelia nevojdú na jednu obrazovku. Animácia je popísaná v kóde číslo 17.

8 Testy

V tejto kapitole sú zobrazené testy presnosti zobrazovania kinectu. Ide o sledovanie vytýčenej trasy, ktorá je vykreslená na obrazovke. Sníma sa pravá dlaň a ľavá dlaň, ktorou pohybujeme čo najpresnejšie po vyznačenej trase.

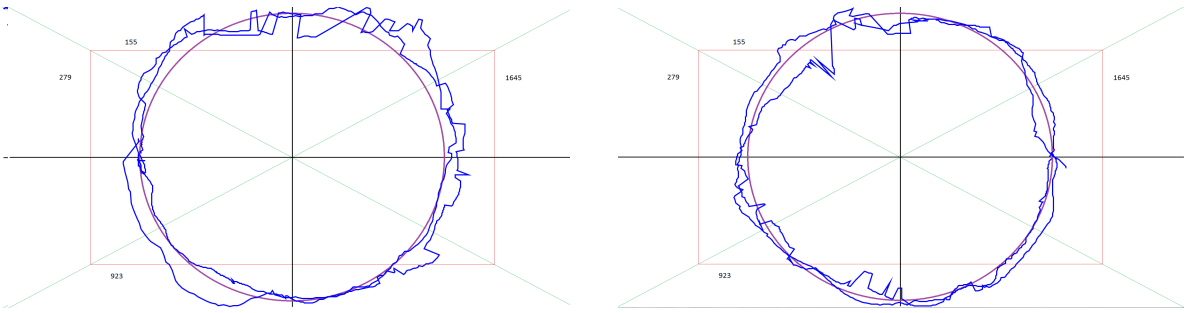
Snímané sú štyri útvary a to obdĺžnik, kruh a dve čiary. Pohyb prvej čiary ide z ľavého horného rohu do pravého dolného rohu. Pri druhej čiare sa sníma pohyb z ľavého horného rohu do pravého dolného rohu. Každý z týchto útvarov je snímaný pravou a ľavou rukou. Výsledné nasnímané body sú znázornené na obrázku číslo 14. Všetky hodnoty, ktoré som získal testami sú pribalené v prílohe.



Obr. 14: Graficky znázornené nasnímané body.

Na obrázku 14 je vidno, že kinect má problémy z plynulým snímaním bodov. Samozrejme plynulosť je ovplyvnená aj osobou, ktorá je snímaná. Nie každému sa podarí nakresliť rukou dokonalý kruh. Na obrázku 15 sú graficky zobrazené body, ktoré sú snímané ľavou a pravou rukou.

Môžeme si všimnúť, že v prípade ľavej ruky je ľavý polkruh hladší a v prípade pravej ruky naopak. Dá sa povedať že ma menší rozptyl snímaných bodov. Je to spôsobené tým že keď ruka prechádza cez stred tela kinect stráca spojenie s niektorými bodmi, alebo si ich mýli s inými. Napríklad ak dáme ruky k sebe tak vykreslí často iba jeden s týchto dvoch bodov.



Obr. 15: Grafický znázornené body kruhu. Snímane ľavou rukou v ľavo a pravou rukou v pravo.

Môžeme vidieť že nasnímané body celkom rozhádzané a nenadväzujú plynulo na seba. To môže spôsobovať problémy pri snímaní rôznych gest. A zároveň pri používaní nevyzerá dobre, keď zobrazené ruky, alebo hocijaký iný zobrazovací bod kinectu skáče po obrazovke. Tomuto sa dá zabrániť viacerými spôsobmi.

Najjednoduchším spôsobom je snímať iba, každý n-tý bod, ako je zobrazené na obrázku číslo 16. V takomto prípade sa ale nezbavíme sekajúceho panáčika, respektíve zobrazovaných bodov. Na obrázku sú vykreslené postupne, každý bod potom každý piaty, desiaty a každý tridsiaty bod. Môžeme si všimnúť že prechody sú plynulejšie, pretože je obrovské množstvo bodov vynechaných, ale aj tak sa nemusíme zbaviť bodov, ktoré sú úplne mimo a narúšajú nám tak obraz.

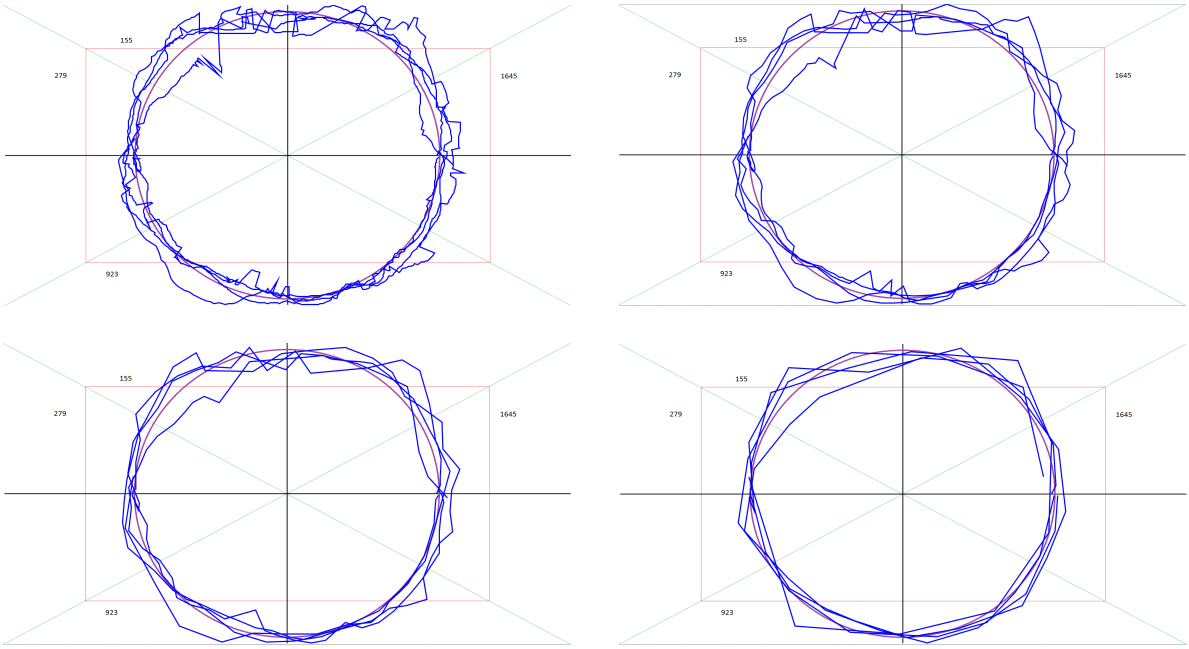
Efektívnejším spôsobom by bolo vytvoriť algoritmus, ktorý by pri snímaní nezobrazoval body, ktoré sú moc ďaleko od bodu, ktorý bol nasnímaný pred nim. pri takejto metóde môže však nastať problém s kinectom, pretože kinect pri rýchlych pohyboch nedokáže nasnímať dostatočné množstvo bodov.

Rýchlosť snímania bodov je zobrazená v tabuľke číslo 1. Ako vstupný bod kinectu je použitá prvá ruka. Gesto je pohyb pravej ruky z pravej strany do ľavej po priamke ako je zobrazené na obrázku 17.

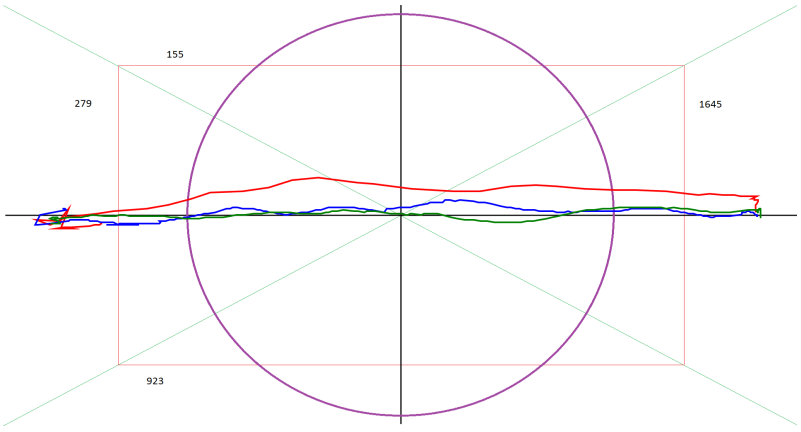
Z výsledkov z tabuľky 1 si môžeme všimnúť že pri rýchlom pohybe sa na vzdialenosti približné 1600 pixelov nasnímalo iba 70 bodov čo môže byť v niektorých prípadoch málo.

Rýchlosť pohybu	**Čas /s**	**Počet bodov**
Pomalý	8,605	242
Rýchlejší	5,436	161
Najrýchlejší	2,334	70

Tabuľka 1: Rýchlosť snímania bodov



Obr. 16: Graficky znázornené nasnímané body kruhu a ich programová úprava



Obr. 17: Grafický zobrazené body k testovaniu rýslosti

9 Záver

Kinect je veľmi silný nástoj s ktorým sa dajú vytvárať jednoducho ovládateľné aplikácie. Dajú sa sním urobiť jednoduché aj veľmi komplikované gestá pre ovládanie užívateľského rozhrania. Existuje obrovská základňa programátorov a veľa tutoriálov pre kinect, takže je s nim jednoduché začať. Nevýhoda kinectu je malé rozlíšenie IR senzora ktoré je iba 640x480 pixelov, ale táto nevýhoda by mala zmiznúť s príchodom nového kinectu.

Pri písaní tohto textu som sa inšpiroval vynikajúcou publikáciou Beginning Kinect Programming [1].

Pavol Kubjatko

10 Literatúra

[1] Jarret Webb, James Ashley, *Beginning Kinect Programming with the Microsoft Kinect SDk,* Apress: 2012.

[2] Stackoverflow, *http://stackoverflow.com*